Optimizing Spatial Databases

Anda VELICANU, Ştefan OLARU
Academy of Economic Studies, Bucharest, Romania
anda.velicanu@ie.ase.ro, stefan4@gmail.com

This paper describes the best way to improve the optimization of spatial databases: through spatial indexes. The most commune and utilized spatial indexes are R-tree and Quadtree and they are presented, analyzed and compared in this paper. Also there are given a few examples of queries that run in Oracle Spatial and are being supported by an R-tree spatial index. Spatial databases offer special features that can be very helpful when needing to represent such data. But in terms of storage and time costs, spatial data can require a lot of resources. This is why optimizing the database is one of the most important aspects when working with large volumes of data.
Keywords: *Spatial Database, Spatial Index, R-tree, Quadtree, Optimization*

1 Introduction

Spatial objects [15] consisting of lines, surfaces, volumes and higher dimensions objects are frequently used in applications of computer-aided design, cartography, geographic information systems. They are described through spatial attributes (length, configuration, perimeter, area, volume, etc.) and also through non-spatial attributes (generation's data, owner, membership of a superior structure, etc.). The values of the objects' spatial attributes represent the *spatial data*.

Spatial data can be divided in point data and regional data. The point data is a point which is completely characterized by its location in a multidimensional space. It may come directly from measurements or by transforming in order to be more easily stored and retrieved [16]. The representation of spatial data in Oracle is done according to the ANSI standard. Spatial database is a collection of spatial and non-spatial data that is interrelated, of data descriptions and links between data. Such a database is optimized so as to best store and interrogate data objects located spatially, as points, lines or polygons. Compared with normal databases, which work only with numeric, character or calendar data, spatial databases offer additional functions that allow processing spatial data types.

Spatial databases are used in areas where aspects of maps or photography are very important.

Optimizing spatial databases means, in the first place, optimizing the queries, which requires less time spent by running the queries before receiving an answer. This is called the response time of a query. The response time can be controlled, minimized by using appropriate indexes depending on the data that is been retrieved. When a query works with one or many rows that store spatial data, it means a spatial index should be used.

Spatial indexes include Grid index, Z-order, Quadtree, Octree, UB-tree, R-tree, kd-tree, M-tree. R-tree and Quadtree are intensively used in combination with spatial data and Oracle Spatial is one example of DBMS (DataBase Management System) that uses automatically these types of spatial indexes for its spatial queries (queries that mostly work with spatial data).

Indexing spatial data [15] is a mechanism to decrease the number of searches, and a spatial index (considered logic) is used to locate objects in the same area of data (window query) or from different locations (spatial junction). Oracle Spatial uses two types of indexing:

R-Tree (SDO_INDEX_TABLE table, maintaining the SDO_RTREE_SEQ_NAME sequence in the virtual table USER_SDO_INDEX_METADATA) and QuadTree (a tree structure, whose nodes have up to four children and is used to divide two-dimensional space, by recursively subdividing itself in four regions) [14].

The distance between two spatial objects is the minimum distance between any points within them. Two objects are within a certain distance of each other if their distance is less than the specified distance. To determine spatial relations, Oracle Spatial has several methods of secondary filtering:

- SDO_RELATE operator evaluates topological criteria;
- SDO_WITHIN_DISTANCE operator determines if two spatial objects are within a certain distance;
- SDO_NN operator identifies the nearest neighbor of a spatial object.

2 Spatial indexing structures

Spatial Indexing is used by spatial databases in order to optimize spatial queries, because indexes used by non-spatial databases cannot handle such operations. Therefore the spatial indexing structure chosen by a DBMS is the one that determines the execution time of its requests and even the ability to return specific results.

The most common spatial indexes are: Grid index, Z-order, Quadtree, Octree, UB-tree, R-tree, kd-tree, M-tree and they are briefly discussed below.

Grid index

In terms of spatial indexing, the grid is a specific area which is divided into a series of contiguous cells, that can have unique identifiers, so they can be used as spatial indexes. Such grids exist in a variety of forms: square, triangular, rectangular, hexagonal, diamond cells etc.

Z-order

Because of its behavior that can be locally preserved, this kind of index is used for data structures to map multidimensional data in one dimension. Once the data is sorted in such a one-dimensional ordering, it can be used as binary search trees, B trees, lists or hash tables. The obtained ranking can be described as equivalent to the ordering that can be obtained by crossing in depth a Quadtree (tree with four dimensions). Because of this connection between the two types of indexation, when using Z-order one can built effectively Quadtrees and multidimensional data structures.

Quadtree

The structure from which this type of indexing starts is a tree whose inner nodes have up to four children. Quadtrees are commonly used to partition a two-dimensional space by dividing it into four identically shaped regions: squares, rectangles etc.

Octree

Are similar to Quadtree, except that the nodes of the tree structure have up to eight children and Octrees are commonly used to partition a three dimensional space by dividing it into eight regions.

UB-tree

UB trees are balanced trees for efficient storage and query of multidimensional data. They are actually B + trees (with information only in the leaves), with records stored as in Z-order.

R-tree

Is the type most common indexing for spatial data. Objects (geometric shapes, lines or points) are grouped using a MBR (Minimum Bounding Rectangle). Objects are added to an MBR with an index, leading to the smallest distance possible.

kd-tree

Kd (k-dimensional) Tree is a space partitioning data structure for organizing points in a k-dimensional space. Kd-tree is used in applications involving multi-dimensional search key.

m-tree

m-tree index can be used for efficient handling of complex object queries using an arbitrary metric.

Since R-Tree and QuadTree indexing are mostly used for spatial data, they will be analyzed further and compared in the following paragraphs.

3 R-tree index

R-tree based indexing requires maintenance of logical tree structure that is implemented as a table. Each R-tree node corresponds to a row in the table and a child pointer in the R-tree corresponds to a primary key of the table's row child. The root pointer for the R-

tree is stored in index metadata and allows browsing from the tree root to leaf nodes. A R-tree's leaf nodes store an MBR area for each data geometry and the geometry identifier. Queries and updates get the R-tree's root and browses down to the leaves. More details about such implementation can be obtained from the source [8].
Whenever a R-tree node is visited, the corresponding spatial row index is selected using the internal SQL command. This means that the query and the update processes for R-trees involve more recursive SQL statements then the Quadtrees. As a result, some operations of data manipulation language (DML), especially updates, are likely to be more expensive [9].
In figure 1 there can be seen an example from source [12], which indicates two different R-trees for the same set of data. R-tree is suitable for storage on secondary sources of memory (eg disk). Each node of the tree is placed on different pages from the disk. This makes the R-tree to be used in particular for applications with large volumes of data, where indexes become too large to fit in main memory.

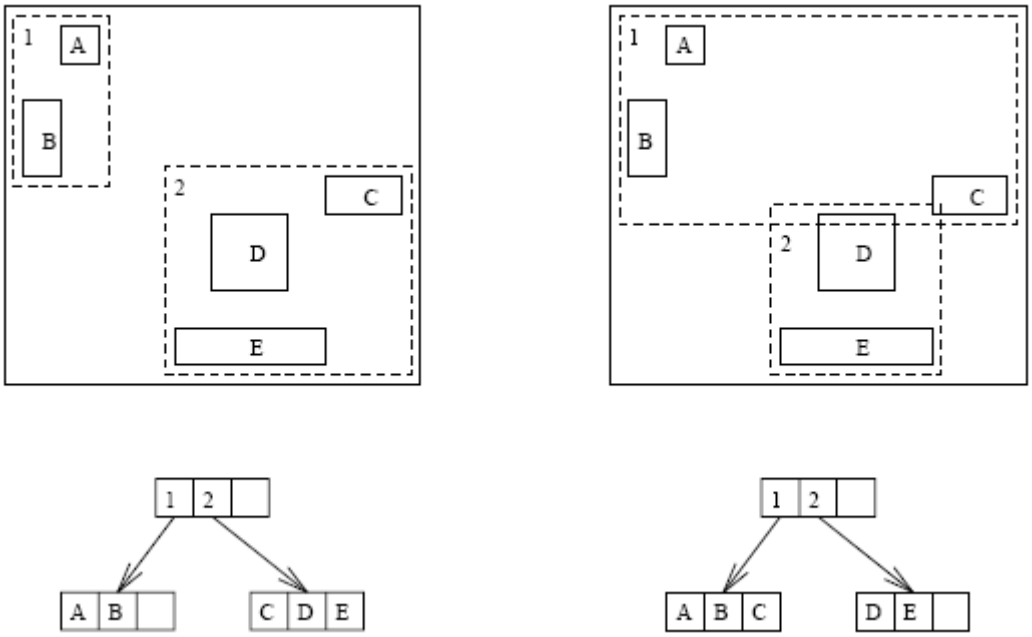

Fig. 1. R-trees for the same data set [12]

When evaluating an index, the most important thing is the response time of a query, if there was applied in advance such an index. The response time is in fact the elapsed time between launching the query and receiving the proper answer. There are several criteria that may affect the response time of an R-tree configuration for the two-dimensional case, presented in [6]:

1. The MBR area
2. The MBR perimeter
3. The distance between bounding rectangles
4. Using the storage space

In multidimensional space, the surface is replaced by volume and the perimeter of a polygon or hyper-polygon is defined by the sum of its extensions in different sizes or by the sum of volumes of the polygon's sides. Coverage polygons with minimum volumes are desirable, so there will be as little unused space (space that is indexed, but contains no data). In a R-tree configuration as less unused space, as more queries that do not index data are likely to be removed from the R-tree, reducing the number of visits on the disc.
Minimizing the 2nd and 3rd criteria and maximizing the 4th criteria will reduce the number of times the disc is accessed. In some

cases these improvements can be contradictory (explanation given in [6]). For example an increase in the use of storage space may result in an increased indexed surface [12].
Building an R-tree index depends on two characteristics:

- The way the objects are inserted in the tree
- Dimensionality

Inserting objects can be made incrementally or in batches. The incremental way implies inserting the objects one by one in the tree, starting from the root and seeking a way to reach a leaf, where the object will actually be added.
Inserting in batches is made by using a data set and building a tree bottom up, starting from the leaf to the root.
The dimensionality of an R-tree can be D-dimensional or linear. In the first case grouping data and coverage polygons in the tree is done by clustering in multidimensional space.
These four methods to build an R-tree (incremental method, batches method, D-dimensional method and linear method) are shown in table 1.

Table 1. Methods for building R-tree indexes [12]

	Incremental method	Batches method
D-dimensional method	Quadric/linear R [10], R+ [5], R* [6]	Optimized iterative R-tree
Linear method	Hilbert R-tree [3]	R-tree by space filling curves [11], [4] Parallel R-tree [13]

On one hand, incremental methods are primarily used to enable working with dynamic data. But if the data set is known from the beginning, batches methods are preferred. This happens because incremental methods are dependent on the way the data set is inserted in the tree. Objects already inserted are likely to be part of the R-tree coverage polygons and is not properly to be represented as such throughout the dataset. However incremental methods are encountered when inserting objects one by one is needed.
On the other hand, batch methods can benefit from the fact that the entire set of data is known in advance. Such methods lead to parallelism. In the bottom-up approach, different processors can be assigned to different sectors of the data space, in order to group objects located in there.
Linear methods have the advantage they are fast, but because of their struggle to map multidimensional polygons in one dimension, positioning and spatial extensions of polygons are not taken into account. It can be a major disadvantage in working with spatial databases.
Following this analysis of methods to build R-tree indexes, I think for complex modern applications (for example the advanced databases developed SOA architecture), the best type of R-tree would be the combination of batch and multi-dimensional modeled trees. When working with large databases, such as advanced databases, it is important to achieve an appropriate index in order to have data queries with the lowest response time. In particular, spatial databases have key characteristics, such as spatial positioning, which are well represented only by multi-dimensional methods. Since the data set is usually known before the application is designed, the R-tree batch building method can also be chosen.

4 Quadtree index

Quadtree term is used to describe a class of hierarchical data structures whose common property is that they are based on the principle of recursive decomposition of space. Decomposition can be in equal parts on each level (that is, regular polygons) or may depend on incoming data. The number of times, in which the decomposition is made, can be fixed in advance or may be governed by the properties of input data [7].

There are several types of Quadtree indexes, classified by the type of data that is represented (surfaces, points, lines or curves), by the independence of the tree's shape on the order in which data is processed or by the variability of the tree obtained from data processing. The following paragraphs describe the types of trees with four nodes (Quadtree): region, point edge.

Region Quadtree

Is a partition of two-dimensional space made by decomposing the region into four equal quadrants, then sub-quadrants and so on, each leaf node containing data which corresponds to a specific sub-region. Each node in the tree has either four children or none (leaf node).

A Region tree with four sizes (Quadtree) and a depth of *n* can be used for representing an image of $2^n \times 2^n$ pixels, each pixel's value is 0 or 1. Root node represents the entire image region. If in a region there are pixels which are not all 0 or 1, the region will divide in the sub-regions. Each leaf node represents a block of pixels that are either all 0 or all 1.

Another example, for how this type of indexing can be used, is storing temperatures from an area. Each leaf node stores an average temperature from the corresponding sub-region.

If Region Quadtree is used to represent a data set (eg latitude and longitude of tourist attractions), the regions are subdivided until each leaf contains one point.

Point Quadtree

This index is based on binary trees used to represent two-dimensional point data type. The difference between Point Quadtrees and Binary Trees is that Point Quadtrees have more complex nodes that contain more than two pointers (left, right) and information, as it happens in binary trees. Therefore a point Quadtree node tree will contain:

- 4 pointers: NW, NE, SW and SE,
- the key represented in x, y coordinates,
- information.

The tree shape depends on the order in which data is processed.

Edge Quadtree

It is used mostly to store lines and not points and curves are approximated by subdividing the cells in a very fine resolution. This can result as very unbalanced trees, which contradicts the fundamental purpose of indexing. Therefore this tree is rarely used.

Whatever type of Quad tree is used for indexing, there are some common characteristics that all trees have to meet:

- the space is split into cells;
- each cell or group of cells has a maximum capacity, and when it is reached the group of cells splits;
- the tree's dimension and shape depend (strictly or not) on how the new data is inserted.

Quadtree indexes are used for image representation, spatial indexing, efficient collision detection in two-dimensional space, inconsistent data storage (such as formatting information in a spreadsheet or matrix calculations), solving multidimensional fields etc. In paper [1] and [2] there are analyzed Quadtree indexes used for addressing spatial data and queries.

5 Comparing spatial indexes

In paper [9] are compared Quadtree and R-tree indexes with examples in Oracle Spatial. Their implementation and several optimizations are described.

An example is the use of inner approximations for Quadtrees and R-trees. Quadtrees use interior spaces for queries and data geometries. Pieces of space, for each data geometry in the spatial indexed tables, are labelled as interior or border, considering whether or not they are within the geometry. Also, the inner surfaces arising from the execution of a query are also identified. The interior appeared during a query and candidate geometries are used together in the intermediate filter to avoid the secondary filter whenever possible. In contrast, R-tree uses only inner queries. As a result, bypassing the secondary filter is not possible in all cases, but will still be effective in most cases.

As data entries, will be used all geometries in 100 miles radius of midtown Manhattan, New York, namely 23,982 geometries, which

makes the data set particularly relevant for this example.
In table 2 we can see a comparison on the response time (in seconds) for queries made on the chosen dataset. These queries use spatial operations such as interaction, inclusion, touching the edges, equality etc. Observe that the R-tree is faster for the implementation of all these operators in Oracle Spatial and obtains for example 35% higher speed for interaction and 65% more time efficient for the *inside* operator.

Table 2. Comparing the average time obtained for Quadtree and R-tree indexes

Oracle Spatial Operator	Quadtree (s)	R-tree (s)
anyinteract	**0.81**	**0.49**
inside	**0.80**	**0.28**
contains	**0.85**	**0.04**
touch	**1.52**	**1.13**
coveredby	**0.88**	**0.66**
covers	**0.44**	**0.05**
equal	**1.53**	**0.04**
overlapbydisjoint	**1.77**	**1.41**
overlapbyintersect	**1.53**	**1.41**

After these comparisons it can be said that although Quadtree has its advantages in terms of more complex types of queries, basic spatial operations are performed much faster using an R-tree indexing type. In Oracle Spatial, MDSYS.SPATIAL_INDEX is the default R-tree index type that gives support for spatial query optimization. Most DBMS also use these type of indexes because overall they had similar or better performance than Quadtrees.

6 Oracle Spatial examples

Oracle Spatial is a component of Oracle Database. It allows users to manage regional and geographic data in a native data type in the Oracle database. Thus, there can be developed a wide range of applications that may include: automated mapping, management facilities, geographic information systems or wireless location services [15].
Oracle Spatial supports the object-relational model for representing the geometry. This model stores an entire geometry in the native Oracle's spatial data type for vector data - SDO_GEOMETRY. An Oracle table can contain one or more columns of SDO_GEOMETRY type. The object-relational model corresponds to a SQL implementation with geometric data, spatial characteristics of tables with the OpenGIS specifications (ODBC / SQL) for geospatial features [15].
In order to see what operations can be affected by spatial indexes in Oracle, and to work with spatial data and spatial indexes, one example will be given. The steps in building one such application will also be described. It's about one table STORES that is described below. We consider a chain store of a company.

➔ STORES:
- *store_id* (C,25) – primary key that unique identifies the store;
- *name* (C,25) – the store's name;
- *address VARCHAR2(50)* – the store's address;
- *geom_shape* (SDO_GEOMETRY) – the geometric shape the store has and its spatial location in the field of activity.

In figure 2 there are examples of some stores that possibly the company takes into account. These are relative positions to the area considered (defined by the frame). If the company works with clients only in Bucharest, the location of stores will be relative to the total area of Bucharest; if the company works with clients from Romania, the location of stores will be relative to the total area of Romania etc. It is also possible that some stores have tangent points or intersect in the space consi-

dered. This thing is possible because of the situation where the company works with both the hypermarket, which holds a store, and the store by itself (it would be if a store is included in another). It's also the situation when a shop hosts within a part of another. These are examples of contact or intersection between two stores.

We consider four stores described by the following geometric properties:

1. A shop with a rectangle shape with extreme points (5,2), (9,2), (9,8), (5,8). For rectangles, Oracle Spatial needs to store two points representing the bottom-left and top-right points above. In this case will store: (5,2), (9,8).
2. A shop with a polygon shape with extreme points (1,3), (4,2), (3,5), (1,5). For polygons, Oracle Spatial needs to store five points representing the extreme points starting from the bottom-left point in counter clockwise order and once again the initial starting point. In this case will store: (1,3), (4,2), (3,5), (1,5), (1,3).
3. A shop in the form of a circle with origin (7,12) and diameter of 4 cm. For circles, Oracle Spatial needs to store three points representing the extreme bottom, right and left points on the circle. In this case will store: (7,10), (9,12), (7,14).
4. A shop with a square shape with extreme points (6,11), (7,11), (7,12), (6,12). For squares, Oracle Spatial proceeds as for rectangles, so it needs to store two points representing the bottom-left and top-right extreme points. In this case will store: (6,11), (7,12).

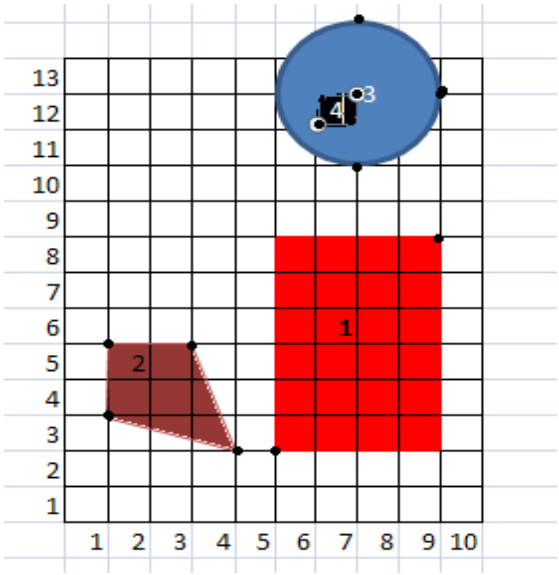

Fig. 2. An example of locating the stores

Will work with STORES table and especially with the geom_shape attribute, in which spatial data is stored.

For solving the problem exposed, we need to follow the next steps:

1. Creating the STORES table

```
CREATE TABLE stores (
 store_id VARCHAR2(25) PRIMARY KEY,
  name VARCHAR2(25), address VAR-
CHAR2(50),
  geom_shape SDO_GEOMETRY);
```

2. Inserting records in STORES table

For the SDO_GEOMETRY row there are many parameters that have the following signification:

- SDO_ELEM_INFO_ARRAY also has 3 parameters - SDO_STARTING_OFFSET (the offset from which the storage in the SDO_ORDINATE vector starts; the first offset is 1 and not 0), SDO_ETYPE, SDO_INTERPRETATION. The last 2 parametres indicate the geometry type.
- SDO_ORDINATE_ARRAY has a variable number of parameters, that represent

the extreme points that can fully describe each type of geometry.

```
INSERT INTO stores VALUES(
  '1',
  'Libelula','Piata Amzei nr.2',
  SDO_GEOMETRY(
    2003,  -- bi-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- a
rectangle (1003 = exterior)
    SDO_ORDINATE_ARRAY(5,2, 9,8) – it
takes only 2 points to define a rectan-
gle: the extreme bottom-left and up-
right
  ));

INSERT INTO stores VALUES(
  '2',
  'Nufaru','Calea Mosilor nr. 270',
  SDO_GEOMETRY(
    2003,  -- bi-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,1), -- a
polygon
    SDO_ORDINATE_ARRAY(1,3, 4,2, 3,5,
1,5, 1,3)
  ));

INSERT INTO stores VALUES(
  '3',
  'Magazin pasaj','Piata universitatii',
  SDO_GEOMETRY(
    2003,
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,4), -- a
circle
    SDO_ORDINATE_ARRAY(7,10, 9,12, 7,14)
  ));

INSERT INTO stores VALUES(
  '4',
  'Farmacia Catena','Piata
universitatii',
  SDO_GEOMETRY(
    2003,  -- bi-dimensional polygon
    NULL,
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3), -- a
rectangle (1003 = exterior)
    SDO_ORDINATE_ARRAY(6,11, 7,12) --it
takes only 2 points to define a rectan-
gle: the extreme bottom-left and up-
right
  ));
```

3. Modifying the USER_SDO_GEOM_METADATA standard metadata view. This operation has to be done before the index is created. The command is run once for each level (for each combination of table-row, in this case for *stores* - *geom_shape*). It contains information about the name of the table that contains spatial data, about the SDO_GEOMETRY row from the table, about the geometry's dimensions and a number (SID) that specifies the coordinate system's value.

```
INSERT INTO user_sdo_geom_metadata
    (TABLE_NAME,
     COLUMN_NAME,
     DIMINFO,
     SRID)
  VALUES (
  'stores',
  'geom_shape',
  SDO_DIM_ARRAY(   -- 20X20 grid
    SDO_DIM_ELEMENT('X', 0, 20, 0.005),
    SDO_DIM_ELEMENT('Y', 0, 20, 0.005)
     ),
  NULL
);
```

4. Creating the spatial index (an R-tree index).

Spatial indexing is a mechanism that helps executing spatial queries in a table based on specific criteria. An R-tree index approximates each geometry with the smallest rectangle that can cover the geometry (it is called MBR – Minimum Bounding Rectangle). For many geometries an R-tree index means hierarchical indexing the MBR rectangles. This type of index is preferred when working with spatial data because it is very quick and works directly on geodesic data.

```
CREATE INDEX stores_spatial_idx
   ON stores(geom_shape)
   INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

5. Executing spatial queries.

- Find the topological intersection between the "1" and "2" stores, respective "3" and "4" stores.

```
SELECT
SDO_GEOM.SDO_INTERSECTION(store1.geom_sh
ape, store2.geom_shape, 0.005) as inter-
sectie
   FROM stores store1, stores store2
   WHERE     store1.store_id='1'     AND
store2.store_id='2';
```

```
INTERSECTIE
------------------------


1 rows selected
```

After running the query we can see that it returns no result, because the stores 1 and 2 are located at distance from one another and they don't intersect.

```
SELECT
SDO_GEOM.SDO_INTERSECTION(store1.geom_sh
ape, store2.geom_shape, 0.005) as inter-
sectie
   FROM stores store1, stores store2
   WHERE     store1.store_id='3'     AND
store2.store_id='4';
```

```
INTERSECTIE
--------------------------------------------------------------------------------------------------------------------------------
MDSYS.SDO_GEOMETRY(2003,null,null,MDSYS.SDO_ELEM_INFO_ARRAY(1,1003,1),MDSYS.SDO_ORDINATE_ARRAY(6,12,6,11,7,11,7,12,6,12))

1 rows selected
```

After running the query we can see that it returns a geometry that as coordinates exactly the extreme points of store 4 and this happens because the store 4 is located inside the store 3.

- Calculate the area of the stores

```
SELECT          m.name          magazin,
SDO_GEOM.SDO_AREA(m.geom_shape,    0.005)
suprafata FROM stores m;
```

MAGAZIN	SUPRAFATA
Libelula	24
Nufaru	6
Magazin pasaj	12.5663706143592
Farmacia Catena	1

4 rows selected

After running the query we can see the name and the area of the stores that asre stored in the database.

- Calculate the distance between the stores with id 1 and 2.

```
SELECT
SDO_GEOM.SDO_DISTANCE(store1.geom_shape,
store2.geom_shape, 0.005) distanta
   FROM stores store1, stores store2
   WHERE     store1.sore_id='1'     and
store2.sore_id ='2';
```

```
DISTANTA
--------------------
1


1 rows selected
```

The query returned the distance 1, between the stores 1 and 2, which is correct, as can be seen in figure 2.

- Determine if there is a spatial relation between the areas of stores 3 and 4.

```
 SELECT
SDO_GEOM.RELATE(store1.geom_shape,
'anyinteract'    ,    store2.geom_shape,
0.005) exista_intercorelare
  FROM stores store1, stores store2
   WHERE     store1.sore_id='3'     and
store2.sore_id ='4';
```

```
EXISTA_INTERCORELARE
--------------------
TRUE


1 rows selected
```

The query that refers to the interaction between the stores 3 and 4 returned TRUE, because the object 4 is in the store 3. The *anyinteract* attribute was used to specify the type of interaction that needs to be checked. The same operation applied to objects 1 and 2 or 3 and 2 would have returned FALSE.

-Determine the intersection area between the stores '3' and '4'

```
select SDO_GEOM.SDO_AREA(
(SELECT
SDO_GEOM.SDO_INTERSECTION(mag1.geom_shap
e, mag2.geom_shape, 0.005)
   FROM stores mag1, stores mag2
   WHERE     mag1.store_id='3'     AND
mag2.store_id='4'),0.005) from dual
```

```
SDO_GEOM.SDO_AREA((SELECTSDO_GEOM.SDO_INTERSECTION(MAG1.
----------------------------------------------------------
1

1 rows selected
```

A complex query was used to calculate the intersection area between 3 and 4 stores. The result is 1, because this is the area of object 4.

7 Conclusions

Spatial databases can be optimized using spatial indexes like R-tree or Quadtree. R-tree approximates each geometry with the smallest rectangle that can cover the geometry (MBR) and is used in Oracle Spatial intensively. In this paper we discussed about spatial indexes and we gave some examples of spatial queries that work with R-tree indexes.

Acknowledgement

This article is a result of the project POSDRU/6/1.5/S/11 „Doctoral Program and PhD Students in the education research and innovation triangle". This project is co funded by European Social Fund through The Sectorial Operational Programme for Human Resources Development 2007-2013, coordinated by The Bucharest Academy of Economic Studies.

References

[1] E. Tanin, A. Harwood and H. Samet, "Using a distributed quadtree index in peer-to-peer networks", *The VLDB Journal — The International Journal on Very Large Data Bases*, Vol. 16, No. 2, April 2007, pp. 165–178.

[2] E. Tanin, A. Harwood and H. Samet, "A Distributed Quadtree Index for Peer-to-Peer," *Proceedings of the 21st International Conference on Data Engineering*, pp. 254–255, 2005, IEEE Computer Society, Washington, DC, USA.

[3] I. Kamel and C. Faloutsos, "*Hilbert* R-tree: An improved R-tree using fractals," *Proceedings VLDB Conference*, 1994.

[4] N. Roussopoulos and D. Leifker, "Direct spatial search on pictorial databases using packed R-trees, *Proceedings ACM SIGMOD*, 1985.

[5] T. Sellis, N. Roussopoulos and C. Faloutsos, "The R+ tree: A dynamic index for multi-dimensional objects, *Proceedings 13th VLDB Conference*, 1987.

[6] N. Beckman, H. P. Kriegel, "The R* tree: An efficient and robust access method for points and rectangles," *Proc. ACM SIGMOD*, pp. 322-331, 1990.

[7] H. Samet, "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys*, Vol. 16, No. 2, June 1984, pp. 187-260.

[8] K. V. Ravi Kanth, S. Ravada, J. Sharma and J. Banerjeen, "Indexing medium-dimensionality data in Oracle," *ACM SIGMOD International Conference on Management of Data Proceedings*, 1999.

[9] R. Kanth, V Kothuri, S. Ravada and D. Abudov, "Quadtree and R-tree indexes in oracle spatial: a comparison using GIS data," *International Conference on Management of Data, Proceedings of the 2002 ACM SIGMOD international conference on Management of data,* Madison, Wisconsin, SESSION: Industrial sessions: commercial implementation techniques, pp.: 546 – 557, 2002.

[10] A. Guttman, "R-trees: A dynamic index structure for spatial searching," *Proc. ACM SIGMOD*, pp. 47-57, 1984.

[11] I. Kamel and C. Faloutsos, "On packing R-trees", *Proceedings 2nd International Conference on Information and Knowledge Management*, pp. 490-499, 1993.

[12] D. M. Gavrila, "R-Tree Index Optimization," *Advances In GIS Research II: Proceedings of the Sixth International Symposium on Spatial Data Handling*, pp. 771–791, 1994.

[13] E. G. Hoel and H. Samet, "Data-parallel R-tree algorithms," *Proceedings 23rd International Conference on Parallel Processing*, pp. 49-53, 1993.

[14] D. Abugov and N. Alexander, "Oracle Spatial User's Guide and Reference, 10g Release 1 (10.1)," *Oracle Corporation*, 2003.

[15] I. Lungu and A. Velicanu, "Spatial Database Technology Used In Developing Geographic Information Systems," *The 9th International Conference on Informatics in Economy – Education, Research & Business Technologies*, Academy of Economic Studies, Bucharest, 7-8 May 2009, pp. 728-734.

[16] M. Velicanu, I. Lungu, M. Muntean and S. Ionescu, *Sisteme de baze de date – Teorie şi practică*, Editura Petrion, Bucharest, 2003, pg. 339.

Anda VELICANU has graduated the Faculty of Economic Cybernetics, Statistics and Informatics of the Bucharest Academy of Economic Studies, in 2008. She is a PhD student in the field of Economic Informatics at the Academy of Economic Studies and since January 2009 she is a Pre-Assistant Lecturer. She teaches Database, Database Management Systems and Economic Informatics seminars at the following faculties: Economic Cybernetics, Statistics and Informatics, Commerce, Marketing and International Business and Economics. Her research activity can be observed in the following achievements: 6 diplomas, 2 scientific awards, 4 proceedings, 3 articles published in scientific reviews, 2 research contracts, 3 books and 1 research grant. She is a member of INFOREC professional association, scientific secretary of "Database support for business" master program and part of the technical team of the "Database Science Journal". Her scientific fields of interest include: Databases, Database Management Systems, Programming, Information Systems.

Ştefan OLARU has graduated the Faculty of Economic Computation and Economic Cybernetics in 2006. He holds a MD degree in Information Security and he is currently a PhD candidate. He is the author of more then 7 journal articles in the field of database security, virtualization and knowledge management. He worked for 3 years with Oracle Corporation from where he gathered important knowledge in database administration. He is an Oracle Certified Associate (OCA) since 2008. He is currently working at Ericsson Romania where he is certified as Senior Managed Services Engineer since 2009.